Информационные системы и технологии

УДК 004.9

М.В. Дидковская, канд. техн. наук, М.В. Донских

Анализ методов поиска в полуструктурированных данных

В работе проведен анализ алгоритмов для получения упорядоченных по релевантности результатов информационного поиска в полуструктурированных данных, выделены недостатки и предложен новый алгоритм с учетом масштабирования.

In the paper the algorithms for obtaining the search results in semi-structured data ordered by relevance are analyzed, disadvantages are marked and new algorithm is proposed taking into account the scale possibilities.

Ключевые слова: *полуструктурированные данные, XML, Dewey Inverted List (DIL), Ranked Dewey Inverted List (RDIL), Algorithm based on Cartesian product (ABCP).*

Введение

На сегодняшний день наблюдается быстрый рост объема данных, доступных в электронном виде. Степень структурированности данных различна. С одной стороны, данные, хранящиеся и обрабатывающиеся с использованием традиционных средств СУБД, имеют строгую и правильную структуру. С другой стороны, аудио и видео изображения можно отнести к полностью неструктурированным данным. Между этими двумя крайностями находится множество данных, называемых полуструктурированными.

Традиционно применяемый в информационно-поисковых системах контекстный поиск, то есть поиск по ключевым словам и без учета структуры анализируемых данных, перестает удовлетворять пользователей своей релевантностью. Особенно существенна данная проблема для Web-поисковых средств, где информационные объемы постоянно возрастают. Таким образом, при поиске должен учитываться не только контекст, но и структура документов.

Целью данной работы является исследование и сравнительный анализ существующих алгоритмов поиска в полуструктурированных данных, выявление недостатков и предложение вариантов масштабирования, рассчитанных на возрастающие объемы информации.

1. Полуструктурированные данные. XML

Полуструктурированные данные - это данные, которые обладают некоторой структурой, но не являются жестко структурированными. [1]

Ниже перечислены основные особенности полуструктурированных данных.

Неправильная структура. Одни и те же сущности реального мира могут моделироваться с помощью различных структур и типов данных. Следовательно, при попытке интеграции разнородных источников необходимо либо приводить данные к общей структуре, либо в результате интеграции данные будут иметь неправильную структуру. Приведение к общей структуре при большом количестве источников может быть невозможным или обобщенная структура будет очень сложной, а ее использование неэффективным. Таким образом, необходимо научиться работать с данными, которые имеют неправильную структуру.

Неявная структура. Значительное количество данных имеет некоторую структуру, но в то же время эту структуру сложно выявить.

Частичная структурированность. В некоторых случаях большая часть данных, работу с которыми необходимо автоматизировать, имеет правильную структуру. Тогда для хранения этой части данных используют традиционные системы управления базами данных, и создают методы связи этой части данных с оставшимися данными, которые не удалось структурировать.

Частое изменение структуры. Структура полуструктурированных данных часто меняется, что необходимо учитывать при разработке систем, работающих с такими данными. [2]

Полуструктурированные данные сложно хранить в реляционной базе данных, поскольку в этом случае будет либо много различных таблиц (что означает многочисленные соединения и продолжительное время поиска), либо единственная таблица с множеством пустых колонок. Полуструктурированные данные оптимальнее хранить как XML [3], и они подходят для XML-базы данных. [4].

Основное отличие поиска в XML заключается в том, что документы не атомарны: они являются не узлами графа, а его подграфами. В XML ищут и ранжируют XML элементы, а не целые документы, а близость слов определяется не только положением в тексте, а и положением в дереве, что позволяет при поиске учитывать не только контекст, но и структуру.

2. Задача поиска в XML

Пусть есть набор XML-документов – граф G = (V, CE, HE), где V – вершины: XML-элементы, CE – дуги вложенности, HE – дуги ссылок.

Основная задача состоит в том, чтобы получить упорядоченные по релевантности результаты информационного поиска в наборе XML-документов. Запросом является набор слов, ответом – упорядоченный набор XML-элементов, которые содержат все слова запроса.

При поиске в XML документах можно выделить следующие проблемы:

- насколько глубокий (в смысле от корня в дереве, которое было до добавления дуг-ссылок) XML-элемент возвращать?
- как учесть структуру и ссылок, и самих XML-деревьев?
- как считать близость слов запроса в разных XML-элементах? Слова могут быть близки в тексте, но находиться в разных узлах графа и быть далеки семантически.

Перед поиском необходимо осуществить ранжирование. Рассмотрим это детальнее.

3. Ранжирование

Предположим, что были найдены все XML-элементы, удовлетворяющие запросу. Далее нужно оценить релевантность каждого XML-элемента. Функция релевантности должна обладать следующими свойствами:

- специфичность результатов. Полагаем, что если у элемента почти все слова запроса сосредоточены в одном потомке, то лучше вперёд поставить не сам элемент, а этого потомка.
- близость ключевых слов. Будем исходить из предположения, что слова из запроса должны располагаться недалеко друг от друга в возвращаемом XML-элементе.
- учёт ссылок. Будем считать, что если на элемент ссылаются много других элементов, то он вероятнее важнее и более удовлетворит ищущего.

Для начала вычислим ElemRank(v) – мера важности XML-элемента, вычисляемая на основе структуры гиперссылок [5].

$$ElemRank(v) = (1-\alpha-\beta-\gamma)\cdot\frac{1}{n} + \\ +\alpha\cdot\sum_{(u,v)\in HE}\frac{ElemRank(u)}{n_h(u)} + \\ +\beta\cdot\sum_{(u,v)\in CE}\frac{ElemRank(u)}{n_c(u)} + \\ +\gamma\cdot\sum_{(u,v)\in CE^{-1}}ElemRank(u),$$

где n - количество XML элементов; $n_c(u)$ - количество потомков у элемента u; $n_h(u)$ - количество ссылок из элемента u; CE – дуги вложенности, HE – дуги ссылок, $CE^{-1}-\{(v,u)|(u,v)\in CE\}$ - дуги от потомков к родителям; α - вероятность перехода из v по ссылке; β - вероятность перехода из v в потомка; γ - вероятность перехода из v в родителя; $1-\alpha-\beta-\gamma$ - вероятность случайного скачка.

Таблица 1. Зависимость времени работы существующей и предложенной процедуры ElemRank от количества вложенных XML-элементов

Время ответа усовершенствованной процедуры для подсчета ранга	Количество вложенных элементов в XML-файле	Время ответа существующей процедуры для подсчета ранга
0,006999969	7	no response (OverflowError)
0,004999876	6	0,019000053
0,003999949	5	0,011000156
0,003000021	4	0,005000114
0,003999949	3	0,003000021
0,002000093	2	0,001000166
0,000999928	1	0,000999928

Таблица 2. Сравнение существующей и усовершенствованной процедуры ElemRank

Запрос	Время работы процедуры Calculate_rank	Время работы усовершенствованной процедуры Calculate_rank	Соотношение полученных значений
system analysis	545,04	164,84	3,31
system	484,34	166,84	2,90
was	483,08	164,84	2,93
Richard	486,34	166,03	2,93
Richard magic Zedd	485,53	163,8	2,96

Заметим, что учитываются не только переходы по гиперссылкам, но и структура XML документа, что значительно усложняет подсчеты. Экспериментальное исследование показало, что максимальное количество вложенных узлов, при котором подсчет происходит быстро и не требует больших затрат памяти, равно 6 (табл. 1). Если количество вложенных узлов увеличить, возникает OverflowError. Во избежание данной проблемы предлагается ввести нормирование. На каждой итерации все значения делятся на значение максимального элемента, благодаря чему сдерживается рост суммы квадратов, а скорость, с которой производятся вычисления, возрастает в 3 раза (табл. 2).

Проанализируем существующие алгоритмы информационного поиска в наборе XML-документов.

4. Анализ алгоритмов для нахождения упорядоченного набора XML-элементов, которые содержат все слова запроса.

4.1. Наивный подход

Обращаться с каждым XML-элементом как с отдельным документом. Данный подход имеет ряд недостатков: требуется много памяти, результаты поиска будут повторяться, не учитывается специфичность.

4.2. Dewey Inverted List (DIL)

Dewey ID – набор чисел, однозначно задающий узел дерева, где каждое число – номер потомка.

Хранится список DIL в виде списка кортежей (DeweyId, ElemRank, PosList), отсортированный по DeweyId, где PosList – это список позиций, на которых встречается слово, для которого создан список.

Рассмотрим работу этого алгоритма. Для каждого списка хранится указатель на обрабатываемый элемент. Из рассматриваемых элементов (под указателями) на данном шаге выбирается минимальный. Далее он сравнивается с последним элементом результирующего списка. Если у них есть общий предок – он записывается в результирующий список вместо бывшего значения. Если нет общего предка – новый элемент добавляется в результирующий список. Увеличивается указатель под обработанным элементом на единицу. Если элементы в списках ещё есть, процесс повторяется с момента выбора минимального элемента. Если нет – заканчивается выполнение. В результате получается искомый список «пересечения» (иногда с нахождением общего предка) списков ключевых слов.

Недостаток: если пользователю нужны только самые релевантные ответы, то всё равно придётся вычислять весь список ответов, а он может быть очень длинным и его формирование потребует существенных ресурсов. [5]

4.3. Ranked Dewey Inverted List

Алгоритм Ranked Dewey Inverted List является модификацией DIL, с той разницей, что сортировка осуществляется не по DeweyId, а по ElemRank. Дополнительно хранится B+ дерево по DeweyId.

Работа алгоритма базируется на спуске по одному из списков от начала до конца, а в остальных, используя структуру дерева, ищется наибольший общий предок для данного элемента. [5]

Недостаток: хотя в среднем показатели данного алгоритма лучше, чем у DIL, при слабой корреляции слов, т.е. когда они встречаются в разных элементах, результаты оказываются значительно хуже – приходится всё равно просматривать почти все списки, что в данном алгоритме существенно дольше простого пересечения DIL. Также недостатком является то, что алгоритм ищет только те элементы, которые содержат все слова запроса.

Общим недостатком представленных выше алгоритмов является то, что они не поддаются распараллеливанию, что является существен-

ной проблемой в условиях постоянно возрастающих объемов информации.

4.4. ABCP (Algorithm based on Cartesian product)

С целью решения проблемы распараллеливания предлагается алгоритм ABCP.

Каждому узлу дерева присваивается Dewey ID. Для каждого слова запроса формируется список Dewey ID. Далее берется декартовое произведение списков – список, элементами которого являются всевозможные упорядоченные пары элементов исходных списков.

При формировании данного декартового произведения используются ленивые вычисления – концепция в некоторых языках программирования, согласно которой вычисления следует откладывать до тех пор, пока не понадобится их результат. Отложенные вычисления позволяют сократить общий объём вычислений за счёт тех вычислений, результаты которых не будут использованы. Программист может просто описывать зависимости функций друг от друга и не следить за тем, чтобы не осуществлялось «лишних вычислений». Применение данной концепции позволяет значительно сократить объем памяти, требуемый для хранения результата.

В каждом декартовом произведении производится попытка слить элементы, которые в него входят. Если результат данной операции отрицательный, то такой элемент больше не хранится и для рассмотрения берется следующий элемент. После обработки всех декартовых произведений получается список, содержащий результат. Как и в предыдущих алгоритмах, если у данных элементов есть общий предок, то в результирующий список записывается он вместо таких элементов.

Формально алгоритм можно представить следующим образом.

1. Формируются списки L1, L2, … , Ln, которые содержат DeweyID для каждого слова из запроса, где n – количество слов в запросе.

2. Инициализируется результирующий массив res.

3. Создается итератор декартового произведения. Декартовое произведение L1 × L2 × … ×Ln списков L1, L2, … Ln есть такое множество L1 × L2 × … ×Ln, элементами которого являются упорядоченные пары (l1, l2, …, ln) для всевозможных $l1 \in L1$, $l2 \in L2$, … , $ln \in Ln$.

4. Для каждых двух элементов декартового произведения находится их общий предок (elem).

5. Если такого предка нет - переходим на следующую итерацию.

6. Если такой предок имеется, в res ищется элемент, который имеет общего предка с elem - old_elem.

7. Если old_elem нет, elem добавляется в конец res, иначе в res old_elem меняется на их общего предка.

8. Выводится результат – res.

Представленный алгоритм использует чистый функциональный подход, поэтому с помощью принципа map-reduce он хорошо масштабируется.

Работа MapReduce состоит из двух шагов: Map и Reduce.

На Map-шаге происходит предварительная обработка входных данных. Для этого один из компьютеров (называемый главным узлом — master node) получает входные данные задачи, разделяет их на части и передает другим компьютерам (рабочим узлам — worker node) для предварительной обработки. Название данный шаг получил от одноименной функции высшего порядка.

На Reduce-шаге происходит свёртка предварительно обработанных данных. Главный узел получает ответы от рабочих узлов и на их основе формирует результат — решение задачи, которая изначально формулировалась.

4.5. Сравнительный анализ алгоритмов

В таблице 3 приведен сравнительный анализ алгоритмов, протестированных на выборке XML-файлов. Общий объем выборки 15Mb, слова в поисковых запросах – в разных XML элементах. В первой колонке отображены входящие данные для запроса, во второй – время работы процедуры, которая строит дерево XML-элементов, в третей – время работы усовершенствованной процедуры для подсчета ранга элементов, в четвертой – время работы «наивного поиска», в пятой – время работы DIL, в шестой – время работы RDIL, в седьмой – время работы усовершенствованного RDIL и в восьмой – время работы ABCP.

Общим недостатком всех алгоритмов является то, что нахождение самых релевантных ответов требует пересмотра всех списков элементов, что существенно отражается на продолжительности работы алгоритма.

Анализ показал, что для запросов, состоящих из небольшого количества слов, целесообразнее использовать предложенный алгоритм ABCP. Более того, данный алгоритм не чувствителен к слабой корреляции слов. При большом количестве слов время его работы может ста-

новится большим чем у алгоритма RDIL, но данный недостаток не будет существенным при масштабировании.

Предлагается повысить эффективность существующих алгоритмов следующим образом:

Использовать модифицированную формулу для подсчета ранга XML-элемента;

Таблица 3. Сравнительный анализ продолжительности работы процедур buildTree, Calculate_rank, «наивный поиск», DIL, RDIL, модифицированный RDIL и ABCP

Слова запроса	buildTree	Calculate _rank	«наивный» поиск	DIL	RDIL	Мод. RDIL	ABCP
system analysis	2,37	164,84	19,15	3,49	3,25	3,10	2,98
system	2,37	166,84	16,65	1,66	1,50	1,52	1,32
was	2,39	164,84	23,08	12,17	6,46	6,51	0,89
Richard	2,41	166,03	15,43	0,94	0,96	0,96	0,92
Richard magic Zedd	2,43	163,8	22,34	3,54	2,79	2,89	3,21

Выводы

В работе проведено исследование и сравнительный анализ существующих алгоритмов поиска в полуструктурированных данных. Основными недостатками является чувствительность алгоритмов к слабой корреляции между поисковыми словами и пересмотр всех списков элементов, что существенно отражается на продолжительности работы алгоритма.

С целью повышения эффективности работы алгоритмов предложено использовать модифицированную формулу для подсчета ранга XML-элемента.

В работе предложен алгоритм ABCP, который не чувствителен к слабой корреляции, работает значительно быстрее существующих алгоритмов на запросах, которые состоят из небольшого количества слов, а также легко распараллеливается, за счет чего в промышленных условиях может работать быстрее и эффективно справляться с большим количеством слов в запросе.

Литература

1. Views for Semistructured Data [Text] : Proceedings of the 24rd International Conference on Very Large Data Bases / S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos et. al.- San Francisco : Morgan Kaufmann, 1997.- 83-90 p.
2. S. Abiteboul. Data on the web: from relations to semistructured data and XML [Text] / S. Abiteboul, P. Buneman, D. Suciu.- San Francisco : Morgan Kaufmann, 2000.- 153-192 p.
3. Wrapper Generation for Semi-structured Internet Sources [Text] : Proceedings of the SIGMOD International Conference on Management of Data / N. Ashish, C. A. Knoblock.- New York : ACM SIGMOD Record, 1997.- 8-15 p.
4. M. Birbeck. Professional XML [Text] / M. Birbeck, Jon Duckett, Oli Gauti Gudmundsson, et. al.- Chicago: Wrox Press, 2001.- 356-382 p.
5. XRANK: Ranked Keyword Search over XML Documents [Text] : Proceedings of the SIGMOD International Conference on Management of Data / L. Guo, F. Shao, C. Botev, J. Shanmugasundaram.- New York : ACM SIGMOD Record, 2003.- 16-27 p.

Национальный технический университет Украины «Киевский политехнический институт»

Поступила в редакцию 26 мая 2012 г.